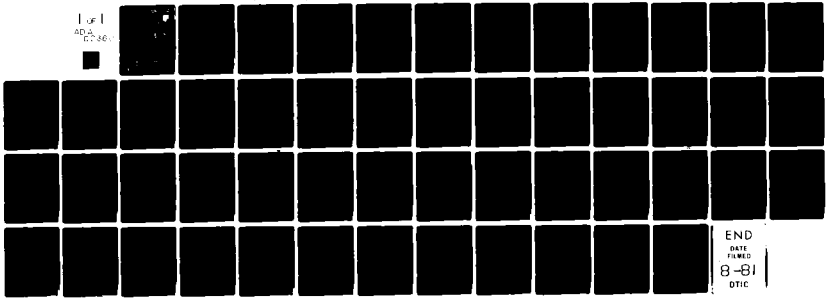
AD-A102 360 PATTERN ANALYSIS AND RECOGNITION CORP ROME NY F/G 9/2
DATA ENTRY FOR C2 PROBLEMS.(U)
JUN 81 T V EDWARDS, D D FERRIS, K L NESTER F30602-80-C-0077
UNCLASSIFIED PAR-81-15 RADC-TR-81-119 NL



END
DATE FILMED
8-81
DTIC

LEVEL II

12

RADC-TR-81-119
Final Technical Report
June 1981

DATA ENTRY FOR C ^{2} PROBLEMS

Pattern Analysis & Recognition Corporation

Thomas V. Edwards
David D. Ferris
Kathy L. Nester
Robert E. Bozek

This effort was funded totally by the Laboratory Directors' Fund.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-81-119 has been reviewed and is approved for publication.

APPROVED: Michael Heffron

MICHAEL HEFFRON
Project Engineer

APPROVED: John N. Entzminger

JOHN N. ENTZMINGER
Technical Director
Intelligence and Reconnaissance Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (IRAA) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: RADC-TR-81-119	2. GOVT ACCESSION NO.: AD-A102 360
3. RECIPIENT'S CATALOG NUMBER:	
4. TITLE (and Subtitle): DATA ENTRY FOR C^2 PROBLEMS	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report, Jun 79 - Feb 81
6. PERFORMING ORG. REPORT NUMBER: PAR Report 81-15	
7. AUTHOR(s): Thomas V. Edwards, Kathy L. Nester, David D. Ferris, Robert E. Bozek	8. CONTRACT OR GRANT NUMBER(s): F30602-80-C-0077
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Pattern Analysis and Recognition Corporation, 228 Liberty Plaza, Rome NY 13440	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 61101F, LD0307C1
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (IRAA), Griffiss AFB NY 13441	12. REPORT DATE: June 1981
	13. NUMBER OF PAGES: 51
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Michael Heffron (IRAA)

This effort was funded totally by the Laboratory Directors' Fund.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Voice Data Entry

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The objective of this effort was to investigate techniques integrating Voice Data Entry using Automatic Speech Recognition equipment and other Automatic Data Entry equipment, such as keyboards and dynamic character pens, etc. The research program focused on methods of combining those Automatic Data Entry Aids to provide high data throughput, low system error rates, and in particular, natural and efficient man-machine interaction.

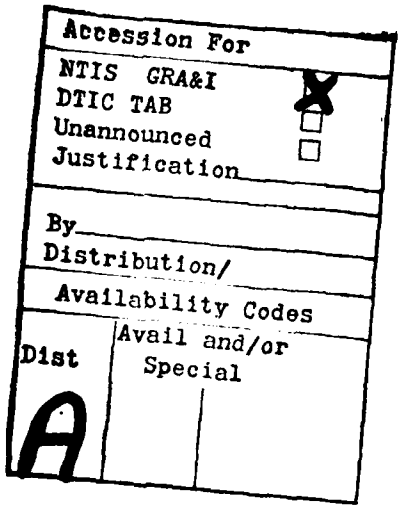
DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

314191

TABLE OF CONTENTS



LIST OF FIGURES

1. INTRODUCTION

Under contract F30602-80-C-0077 entitled "Data Entry for C^2 Problems", PAR Technology Corporation designed, developed and installed an Integrated Data Entry (IDE) System on the RADC SIGINT Support Facility PDP 11/70 computer. Data entry devices currently include an Automatic Speech Recognition (ASR) device, a Drawn Character Recognition (DCR) device and any terminal on the PDP 11/70. The system is also expandable to allow future incorporation of devices such as graphics tablets and new terminals as they become available. The use of more than one data entry device in a data entry task will provide higher data throughput rates than is possible with a single device. This effort consisted of interfacing an ASR to the PDP 11/70 and the development of software to provide data entry device inputs to a user generated program. The system will be used in future RADC in-house data entry experiments.

The ASR is a VIP-100 speaker-dependent isolated word recognition system manufactured by Threshold Technology Inc. It consists of a speech preprocessor which extracts phoneme and spectral based measurements for a spoken utterance and a NOVA 1200 minicomputer which time normalizes the input feature data and correlates the input feature array (FAR) with a reference array (RAR) to determine the word spoken. Libraries are typically 64 to 100 words in size. One of the applications of the IDE system is to enable the switching between stored speech libraries in a timely fashion. To achieve this objective, it was necessary that the speech correlation algorithm be moved to the PDP 11/70. The NOVA was interfaced via an RS232 serial 9600 baud link to the PDP 11/70. When FAR data is available, it is transferred across the line to the PDP 11/70 who then performs the correlation. This has resulted in extremely fast library selection and word recognition rates. Details on the hardware interfacing can be found in the "Interim Technical Report."

Software development resulted in the following modules:

- CREATE UTILITY
- TRAIN UTILITY
- IDE SUBROUTINES

The Create utility is used to define a particular data entry device configuration and the hierarchy of individual devices. It is also used to attach meaning to a set of specific device inputs. This program reads an ASCII source file containing the above information and creates a machine readable system definition file (SDF).

The Train utility is used to facilitate training of the ASR device. Typically, each word to be recognized is spoken 10 times. Train then averages the 10 resulting feature pattern and inserts it into one entry of a library file. Linkage between the SDF, the speech library, the data entry devices and the user program is made extremely simple by the existence of two IDE subroutines. The user need only link his program with these subroutines in order to run the IDE system. One subroutine is called to initialize the system, while another is used to retrieve inputs from the various entry devices and their associated definition from the SDF file.

Acceptance tests were run at RADC to verify operation of the IDE and the speech correlation algorithm in the PDP 11/70. Time between input of speech utterance and recognized word output on the echo device was demonstrated to be less than .5 seconds. The speech classification algorithm was tested with 5 speakers, speaking 50, 3-digit connected sequences. Results were that for all five speakers, 97% of the digits were correct. It was found that one speaker experienced much higher error rates than the other five. The reason for the increased errors is thought to be due to the fact that this particular speaker had not had much previous experience with the ASR device whereas the other

four speakers had used the device on several occasions and thus were very familiar with the manner in which words must be spoken. Disregarding this speaker, the remaining four speakers exhibited a better than 99% correct classification.

In the following sections, we will discuss the Create, Train and IDE tools from a user's point of view. Explicit examples are given whenever possible. It should be noted that although a number of files of complex design are used in the system, the user needs only understand the source (ASCII) version of the SDF file in order to use the software.

2. THE CREATE UTILITY

The purpose of the CREATE task is to interpret and verify the system definition file specified and create an ordered machine readable version of the system definition file for later use by the IDE system. Optionally, the CREATE task will create an ordered formatted listing of the system definition file. The following sections describe in detail the format/syntax of the system definition file, the format of the machine readable system definition file, and the command syntax for the CREATE utility.

2.1 THE SYSTEM DEFINITION FILE

The SYSTEM DEFINITION FILE is the user created file which completely describes the device configuration to be used and interpretation instructions for the data received from the input devices. The device configuration may contain any devices known to the IDE system, including a single device. The interpretation instructions for the set of selected input devices can be thought of as an N-dimensional lookup table, (where N is the number of input devices) which returns a meaning for each unique set of N device inputs. If the system definition file does not contain a meaning for a particular set of N device inputs, the meaning is 'unknown' and the user is informed.

For each system definition, the devices prior to the ASR device are entified as the KEY device. The significance of the KEY DEVICE is that a separate ASR reference library (RAR) corresponding to each KEY entry.

An example of a user-created system definition file is given in Figure 2-1.

TST2.SRC

```
TERM,DRC,ASR
BOATDAT.SRC
CARDAT.SRC
BIKEDAT.SRC
```

CARDAT.SRC

```
TERM,DRC,ASR
C ;CAR   ,C ;CHEVY        ,CAPRICE        =CAPRICE
C        ,C               ,MONZA          =MONZA
C        ,C               ,CAMERO         =CAMERO
C        ,F ;FORD         ,MUSTANG        =MUSTANG
C        ,F               ,GALAXIE        =GALAXIE
C        ,P ;PONTIAC      ,FIREBIRD       =FIREBIRD
C        ,P               ,SUNBIRD        =SUNBIRD
```

BOATDAT.SRC

```
TERM, DRC, ASR
B ;BOAT ,S ;SPEEDBOAT    ,WELLCRAFT      =WELLCRAFT
B       ,S               ,GLASTRON       =GLASTRON
B       ,S               ,WHALER         =WHALER
B       ,C ;CRUISER      ,OWENS          =OWENS
B       ,C               ,LUHERS         =LUHERS
B       ,C               ,VIKING         =VIKING
```

BIKEDAT.SRC

```
TERM,DRC,ASR
A ;BICYCLE,S ;SCHWINN          ,ROADSTAR       =ROADSTAR 10 SPD.
A          ,M ; MONTG. WARD    ,MONKEY         =MONKEY BIKE
```

Figure 2-1 Example of a Three Input Device System Definition File and the Specified Include Files

2.1.1 System Device Declaration

Line one (1) of the system definition file identifies the devices which are to be used in the IDE system configuration. The format of line one (1) is as follows:

line (1) DEV1, DEV2, DEV3, ...,DEVn

The identification of the devices shall be by three (3) character alphanumeric mneumonic, the first character of which must be a letter. The device mneumonic can include any of the following characters: A-Z, a-z, 0-9. All characters, whether upper or lower case will be interpreted by the software system as upper case.

The field separator shall be a single comma (','). The 'space' and 'tab' characters are provided to the user for formatting. Any combination of spaces, tabs, or both may be present anywhere in the line so long as they do not interupt the device mneumonics. All of the device mneumonics will be located on line one (1). There are no comments allowed on line one (1).

2.1.2 Optional File Comment Field

Following line one (1), it is permissible to include a comment field which is only for the purpose of user documentation. Any lines consecutively following line one (1) which have a semi-colon (';') as the first character will be considered user comments and be ignored by the system. A sample system definition file has the following format.

```
line (1):  DCR,ASR
line (2):  ;This is the user comment area.
line (3):  ;Any lines consecutively following
line (4):  ;line 1 which start with a ';' are
line (5):  ;comments.
line (6):  I1 ;comment, I2 ;comment =MEANING
line (7):  I1 ;comment, I2 =MEANING
  *
  *
  *
line (k):    I1 ;comment, I2   =MEANING
line (k+1):  FILESPEC.EXT
line (k+2):  I1 ;comment, I2   =MEANING
```

The user comment area is optional. If present, it will be skipped over by the IDE software.

2.1.3 Definition Lines

Any lines which follow the device specification line (line 1) and the optional user comment area are either 'definition lines' or, file specifications. A 'definition line' has the following form:

```
I1 ;comment, I2 ;comment, ...In ;comment =MEANING
```

where I1 is the input from the first device on line one (1), the key device, and I2, I3, ... In are the inputs from devices 2,3, ...n as listed on line one (1). The input can be any character or string as detected by the particular device.

The field delimited by a semi-colon (';*comment*') is an optional in-line comment area available to the user. There can be comments associated with with each device input. The in-line comment is only for the user's reference and is not used by the system. It is, however, provided to the user on the optional listing produced by the CREATE utility.

The final field on the line, delimited by the equals ('=') contains the are specified. The 'MEANING', as with the inputs, can be any character (string).

Any combination of spaces and tabs may be mixed between the fields described above.

2.1.4 Include Files

In order to eliminate the re-typing of some part of a system definition file which is to be used again, the 'Include File' capability exists. If, anywhere after line one (1) and the optional user comment area, there exists a line which contains a file specification, it will be interpreted as an 'Include File'. The format and syntax of an include file is exactly that of the system definition file. An 'Include File' can reference another 'Include File' to a total of level of nesting equal to (4). The 'Include File' must be the only entry on the line.

The device declaration (line 1) of the 'Include File' must match the system definition file.

2.2 COMMAND LINE FORMAT

The CREATE utility shall use the standard RSX command line format.

CRT> [OBJECT][LIST][/SP]=SOURCE where:

SOURCE is system definition file input

OBJECT is the machine readable output

LIST is the optional list file -

/SP - indicates that the listing, if created, is to be spooled (Default =-SP)

The CREATE utility shall use the MCR GCML routine for input. Command line interpretation shall be via the string manipulation routines of STRLIB.OLB.

2.3 OPTIONAL LISTING

For the system definition file shown, together with the include files, the listing generated by the CREATE task shall be representative of the following page, Figure 2-2.

```
                    INTEGRATED DATA ENTRY SYSTEM

                    *****        TST2.SRC        *****

                    OWNER: [351,1]  6-JUNE-80    10:23

DATA ENTRY DEVICES::   TERM,  DRC,  ASR    ; KEY DEV:: TERM

TERM ENTRY          DRC ENTRY           ASR ENTRY       DEFINITION
----------          ---------           ---------       ----------

A ;BIKE          - M ;MONTG. WARDS      MONKEY          =MONKEY BIKE

                   S ;SCHWINN           ROADSTAR        =ROADSTAR 10 SPD.

B ;BOAT          - C ;CRUISER           OWENS           =OWENS
                                        LUHERS          =LUHERS
                                        VIKING          =VIKING

                   S ;SPEEDBOAT         WELLCRAFT       =WELLCRAFT
                                        GLASTRON        =GLASTRON
                                        WHALER          =WHALER

C ;CAR           - C ;CHEVY             CAPRICE         =CAPRICE
                                        MONZA           =MONZA
                                        CAMERO          =CAMERO

                   F ;FORD              MUSTANG         =MUSTANG
                                        GALAXIE         =GALAXIE

                   P ;PONTIAC           FIREBIRD        =FIREBIRD
                                        SUNBIRD         =SUNBIRD

                    *****       END OF LISTING       *****
```

Figure 2-2 Representative Output Listing Produced by the CREATE Utility for a Three Input Device System Definition File

3. TRAIN UTILITY

When the ASR (Automatic Speech Recognizer) is used as one of the devices in a data entry system, each word that the user wishes to have recognized must be trained. The Train Utility has the responsibility of letting the user train any word he wishes, with a minimum of difficulty. To aid the user, this manual has been written, and wherever possible, explicit examples are used to describe the usage of each option clearly and accurately. This section is broken up into two parts, the first of which will explain exactly how the software sets up the various files. Part two will describe in detail how to run the train utility and the different switches and options involved.

3.1 TRAIN UTILITY FILE STRUCTURE

Before 'TRAIN' can be run, a SDF (System Definition File) is needed. The 'CREATE' task has been designed to accomplish this. A description of how to use this task is described in Section 2 of this manual. Also included in the create task description is a complete description of how the SDF is set up.

Assuming an SDF exists, an undefined and unordered ASR (Automatic Speech Recognizer) file must be created. An undefined ASR Library has the following characteristics: 1) all undefined entries in the characteristics buffer are zero, 2) there is no characteristics index, however two blocks are allocated for this index in each library, 3) the message buffer is unordered, and 4) the message index is set so that it points to each of the ASCII words in the unordered message buffer.

When an ASR File is completely defined, it must be reordered. The reordered ASR file will be the final version of the ASR. The characteristics of an ordered ASR File are as follows: 1) The number of bits set in the binary equivalent of each 32 word voice characteristic is calculated. The entries are then reordered from lowest number of bits set to highest number of bits

set and written back to the characteristics buffer. 2) The characteristics index is calculated and written to the two blocks allocated. 3) The software then reorders all of the ASCII text in the message buffer in the same order that the characteristics have been ordered in. 4) Once the message buffer has been set up, the message index is calculated and written to the proper location. These four steps are done separately to each library and upon completion, the ASR file is ready to be handled by the IDE task.

The ASCII text and pointers in the ASR level of the SDF must also be reordered in the same way that the characteristics buffer is ordered. Together the ordered ASR file and ordered SDF allow the IDE task to make a decision as to exactly what the user said and the associated meaning. After each library is completed, the MFI (Master File Index) is updated. This index points to the beginning block of each library. (See Figure 3-1 for a diagram of the ASR library file structure).

3.1.1 The Master File Index (MFI)

As previously stated, each word in the MFI (see Figure 3-2) points to the starting block of each library. However, the first and last word are reserved for something else. The first word contains the number of libraries in the ASR File. In the unordered ASR file, the last word of the MFI is the total number of entries trained in all of the libraries. When the ASR is reordered, the last word of the MFI is set equal to -100. In this way the software can test the last word and know if the library has been ordered and if not, how many words have been trained.

3.1.2 The Library Index

The Library Index (see Figure 3-3) points to an entry or group of entries in the characteristics buffer with a given number of bits set. The position of the odd bytes indicate how many bits are set in the entry or entries being pointed at by the index. The formula for number of bits set is: (number of

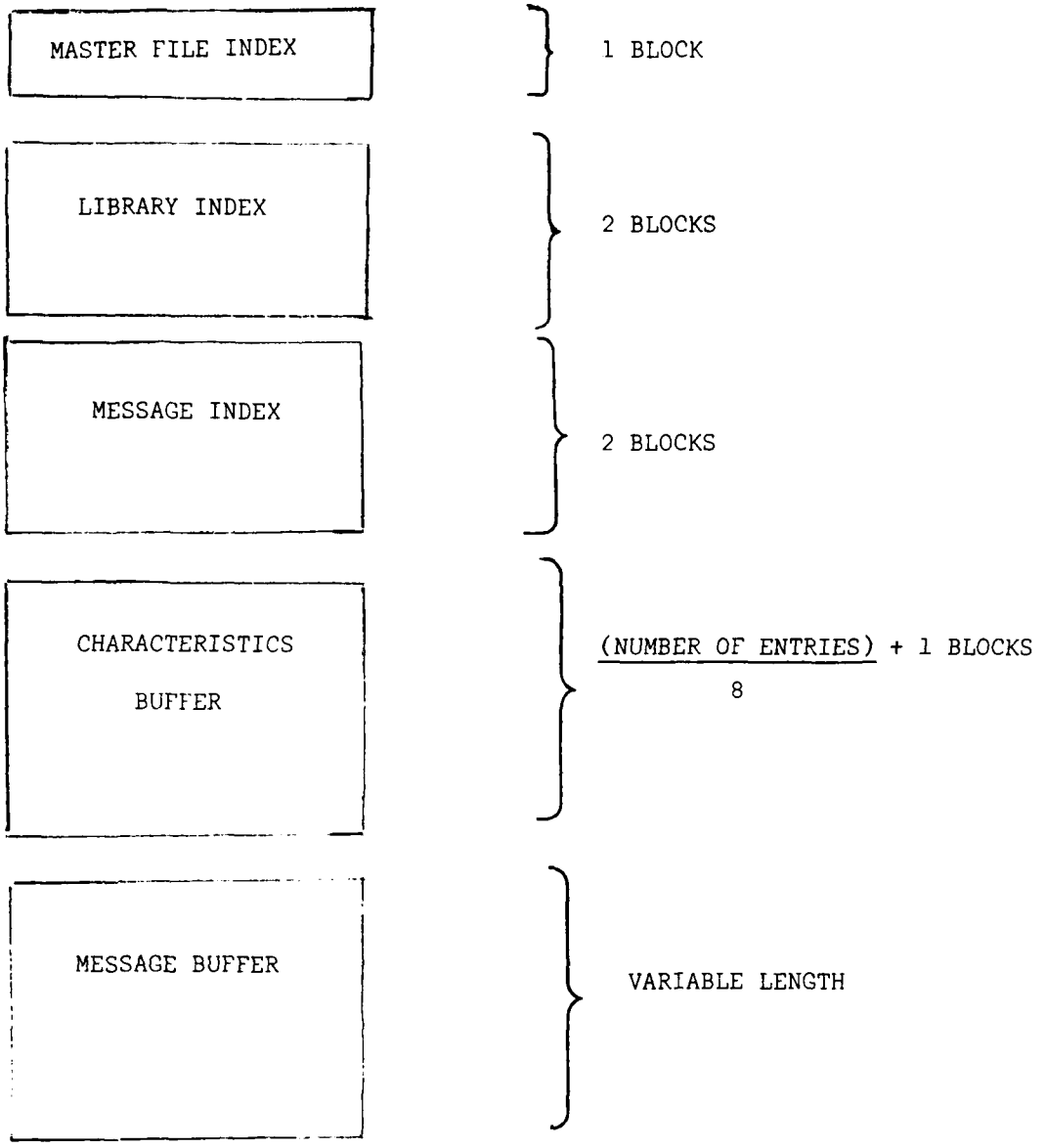
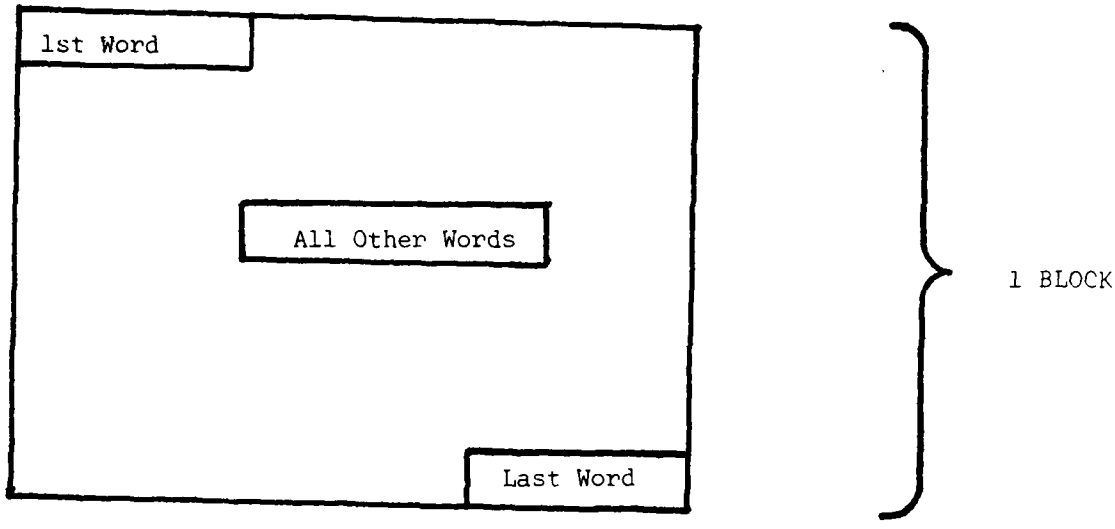


Figure 3-1 An ASR Library File Structure



1st word of MFI: number of libraries in the ASR file
Last (256th) word of MFI is: 1) number of entries trained if ASR file is unordered. 2)-100 is ASR file is ordered.

All other words (2nd-255th): contains the starting block number of each library.

Figure 3-2 Master File Index (MFI) file structure

ODD BYTE (325)	EVEN BYTE (326)
- 120	5

Assume the number of the odd byte is 325

Assume the contents of the odd byte is -120 (decimal)

Assume the contents of the even byte is 5

Number of bits set being pointed at (325 + 1) / 2 = 163

Number of bytes already indexed = (-120 + 128) = 8
(128 is added to the contents of the odd byte as an offset)

Number of entries with 163 bits set = 5

Figure 3-3 Library Index File Structure

1st WORD	2nd WORD	
	low byte	high byte
Block Number	Starting Word in Block	Number of Bytes

Figure 3-4 Message Index File Structure

the odd byte +1)/2, therefore the library index that starts with byte number 325 points to all entries with (325 + 1)/2 or 163 bits set. The contents of the odd byte is the number of entries that have already been indexed while the contents of the even byte is the number of entries with the number of bits set specified by the odd byte. The library index is 512 words long or 1024 bytes long, therefore, 1023 is the highest odd byte and it points to (1023 + 1)/2 or 512 bits set.

3.1.3 The Message Index

The message index (see Figure 3-4) points to each separate ASCII entry in the message buffer. Each entry in this index is 4 bytes long. The entire index requires two blocks or 512 words. The low order word contains the block number that the ASCII entry is in. The low order byte of the high order word contains the word in the given block that the entry starts on and the high order byte of the high order word contains the number of bytes that make up the entry.

3.1.4 Summary

The description of the ASR file structure was intended to give the user a brief summary of the structures of files that the train utility creates and what information is contained in them. Knowledge of the file structure should be helpful in using the Train Utility, however, it should be stressed that the user does not have to be aware of this information. The Train Utility properly sets up the ordered and unordered ASR files and uses the information contained in them without the knowledge of the user.

3.2 TRAIN UTILITY COMMAND LANGUAGE

If the Train Utility is not installed, type in RUN [205,3]TRAIN. This will return a "TRN>" prompt. The user is now in the Train Utility. In response to the prompt, the user has a variety of options. All of these

options will be explained in detail on the following pages. Only the first letter of the option or switch needs to be used.

FILSPC/CREATE (where FILSPC is the name of the SDF).

This option creates an unordered and undefined ASR file using the ASR entries from the SDF. The ASR file created has the same name as its SDF counterpart but with a .ASR extention. This command must be used only once for each SDF.

EXAMPLE:

Directory Before Executing Command	Directory Immediately After Executing Command
EXAMPLE.SDF;1	EXAMPLE.SDF;1 EXAMPLE.ASR;1

In the event the user used this command twice on the same SDF this is what would happen.

Directory After Using Command Once	Directory After Using Command Twice
EXAMPLE.SDF;1 EXAMPLE.ASR;1	EXAMPLE.SDF;1 EXAMPLE.ASR;1 EXAMPLE.ASR;2

Assuming the SDF's are identical, both ASR files will be identical. However, if EXAMPLE.ASR;1 had several trained entries, EXAMPLE.ASR;2 would still not have any trained entries. If the file were purged EXAMPLE.ASR;1 would be deleted.

FILSPC/MODIFY (where FILSPC is an ASR file).

If an ASR file already exists and the user wishes to modify the file in some way, i.e., train some of the entries, he would use this command. This command searches the directory for the proper ASR file and opens up the most recent version. Modify does not create a new version, it just opens up the version that already exists.

It should be noted that a file which has been reordered may not be modified. If the user attempts this, an error message will be echoed.

FILSPC/LIB'N' (where N is a number between 1 and 5).

This command identifies FILSPC as a library available for selective search. This command will be discussed in more detail when some of the other commands are discussed.

SCOPE

When 'Scope' is typed in, the current scope will be printed out. If the scope has not been set, the entire scope will be printed. For a 1 level file there is no scope.

EXAMPLE

Assume that the file in Figure 3-5 is being modified and the scope is desired.

```
                    I N T E G R A T E D   D A T A   E N T R Y   S Y S T E M

                    #####     SY:TST2.SRC    #####

                              DATE:26-JAN-81     TIME:02:16:15

                              DATA ENTRY DEVICES:     TTO,DRC,ASR

 TTO ENTRY      DRC ENTRY      ASR ENTRY    = DEFINITION
 ---------      ---------      ---------    ------------

A;BIKES
                M;WARDS
                                ^C ABORT    = -
                                ^U ERROR    = -
                                ^W FLUSH    = -
                                MONKEY      = MONKEY BIKE
                S;SCHWINN
                                ^C ABORT    = -
                                ^U ERROR    = -
                                ^W FLUSH    = -
                                ROADSTAR    = ROADSTAR 10 SPD.
B;BOAT
                C;CRUISER
                                ^C ABORT    = -
                                ^U ERROR    = -
                                ^W FLUSH    = -
                                LUHERS      = LUHERS
                                OWENS       = OWENS
                                VIKING      = VICKING
                S;SPEEDBOAT
                                ^C ABORT    = -
                                ^U ERROR    = -
                                ^W FLUSH    = -
                                GLASTRON    = GLASTRON
                                WELLCRAFT   = WELLCRAFT
                                WHALER      = WHALER
C;CAR
                C;CHEVY
                                ^C ABORT    = -
                                ^U ERROR    = -
                                ^W FLUSH    = -
                                CAMERO      = CAMERO
                                CAPRICE     = CAPRICE
                                MONZA       = MONZA
                F;FORD
                                ^C ABORT    = -
                                ^U ERROR    = -
                                ^W FLUSH    = -
                                GALAXIE     = GALAXIE
                                MUSTANG     = MUSTANG
                P;PONTIAC
                                ^C ABORT    = -
                                ^U ERROR    = -
                                ^W FLUSH    = -
                                FIREBIRD    = FIREBIRD
                                SUNBIRD     = SUNBIRD
```

Figure 3-5 System Definition File Source

```
TRN>TST2/M or C      ; If the ASR has already been created, the switch
                       M is used.  If not the switch C is used.

TRN>SCOPE            ; Print out the current scope, Since
A-M                  ; the scope was not set to anything
A-S                  ; the entire Scope has been printed.
B-C
B-S
C-C
C-F
C-P
End of Scope
```

SCOPE/KEY: a, b, c...; (where a,b,c... are paths up to but not including the ASR level).

Answering the 'TRN>' prompt with this command sets the Scope to the specified ASR libraries.

EXAMPLE (See Figure 3-5 for TST2)

```
TRN>TST2/M or C

TRN>SCOPE
A-M
A-S
B-C
B-S
C-C
C-F
C-P
End of Scope

TRN>SCOPE/KEY:  C-C          ; SET the key to C-C (car-chevy)
TRN>SCOPE                    ; List out the current Scope
C-C
End of Scope
TRN>SCOPE/KEY:  B-C, C-F
TRN>S
B-C
C-F
End of Scope
TRN>
```

SCOPE/ALL

This command sets the Scope back to the entire SDF.

EXAMPLE (Using 'TST2' in Figure 3-5)

```
TRN>SCOPE        Show current Scope
C-C
End of Scope
TRN>SCOPE/ALL    ; SET SCOPE to everything
TRN>SCOPE
A-M
A-S
B-C
B-S
C-C
C-F
End of Scope
TRN>
```

SCOPE/INCLUDE: a,b,c... (Where a, b, c are paths up to
SCOPE/DELETE: a,b,c... but not including the ASR level).

The include or delete switch allows the user to include in or delete from the current Scope, everything associated with the given path or paths.

EXAMPLE (See Figure 3-5)

```
TRN>SCOPE
A-M
A-S
B-C
B-S
C-C
C-F
C-P
End of Scope

TRN>SCOPE/DELETE:  A
TRN>SCOPE
B-C
B-S
C-C
C-F
C-P
End of Scope
TRN>SCOPE/Include:  A
TRN>SCOPE
A-M
A-S
B-C
B-S
C-C
C-F
C-P
End of Scope
TRN>SCOPE/DELETE:  A-M
TRN>SCOPE
A-S
B-C
B-S
C-C
C-F
C-P
End of Scope
TRN>
```

It should be stated that anything done by the delete command can be undone by the include command and vice versa.

LIST

When LIST is entered without a switch, it will act precisely as if the user typed in SCOPE.

LIST/LIB'N' (where N is 1-5)

LIB'N' is an ASR file that the user declared as a library. When this command is executed the current scope of library LIB'N' will be listed. To declare a file a library, use the command FILSPC/LIB'N'.

LIST/UNDEFINED

Since the user may not completely train a file in one sitting, there is the likelihood that the user would forget which words have not been trained. Using this command, the "TRAIN UTILITY' will list any undefined ASR entries that are within the current SCOPE.

```
EXAMPLE (Using TST2 in Figure 3-5)
TRN > LIST/UNDEFINED
A-M
            ~C
            ~U
            ~W
            MONKEY
A-S
            ~C
            ~U
            ~W
            ROADSTER
B-C
            ~C
            ~U
            ~W
            L.UHERS
            OWENS
            VIKING
B-S
            ~C
            ~U
            ~W
            GLASTRON
            WELLCRAFT
            WHALER
C-C
            ~C
            ~U
            ~W
            CAMERO
            CAPRICE
            MONZA
C-F
            ~C
            ~U
            ~W
            GALAXIE
            MUSTANG
C-P

            ~C
            ~U
            ~W
            FIREBIRD

            SUNBIRD
*END OF SCOPE *
TRN>S/K:B-C,C-P---------->NARROW THE FIELD OF SCOPE
TRN>L/U
```

```
B-C
            ~C
            ~U
            ~W
            LUHERS
            OWENS
            VIKING
C-P
            ~C          Only undefined entries in current Scope are listed.
            ~U          All others are ignored.
            ~W
            FIREBIRD
            SUNBIRD
** END OF SCOPE **
TRN>
```

LIST/KEYS

This command will ignore the current scope and echo to the user all of the keys in the SDF.

```
EXAMPLE
TRN>S/A------------>Scope is set to entire field
TRN>L/K
A-M
A-S
B-C
B-S
C-C
C-F
C-P
   ** END OF SCOPE **
TRN>S/K:A-M,C-F---->Scope field is narrowed
TRN>S
A-M
C-F
   ** END OF SCOPE **
TRN>L/K
A-M
A-S
B-C                 Ignores current scope
B-S
C-C
C-F
C-P
   ** END OF SCOPE **
TRN>
```

TRAIN

Entering this command will place the 'TRAIN' UTILITY' in the training mode. This command looks at the current scope and will only allow the undefined entries in the current scope to be trained. If the user wishes to train words that are out of the scope, he must leave the 'TRAIN' mode and set the Scope using the commands described previously.

TRAIN/KEY: KEYNAME: ASCII INPUT

There is the possibility that a particular word would have been trained improperly i.e., when repeating the word the user says the word incorrectly. If in response to the TRN> prompt the user types this command the user may retrain the word.

EXAMPLE: (Using TST2 in Figure 3-5)

```
TRN>TRAIN/KEY: C-C: CAMERO
CAMERO
```

The user may now respond to the word using the responses described next.

When the user responds to the TRN> prompt with the command 'TRAIN' the word to be trained will be echoed.

EXAMPLE: TRN>TRAIN
('word to be trained')

There are several responses that the user may input.

RESPONSES

'.' At this time the user does not wish to train this word.

<CR> the user wishes to train this word using the speech box. 'TRAIN' will prompt with 'PLEASE REPEAT'. After repeating the word 10 times, the user will be prompted with the next word and again he will be able to enter a particular response.

EXAMPLE

```
TRN>TRAIN
WORD<CR>
PLEASE REPEAT  1
     |
     |
     |
PLEASE REPEAT  10

NEXT WORD (Option)
```

^Z When the user wishes to terminate a training session before everything is defined, this response will exit the training MODE. (Training mode is automatically exited when all entries in the current scope have been trained)

Since it would be cumbersome to have to train each word everytime it is used, the copy option has been included. In this way words that are frequently used, ie. the control characters, have to be trained only once.

COPY/LIB'N' (where LIB'N' is a library).

Using this command the user may use the ASR file declared LIB'N' to train all of the words in the scope of the current ASR file that are also in LIB'N'. Both the current ASR file and LIB'N' must have the same number of levels. The ASR level of both files must also be identical.

EXAMPLE

Assume CNTRL.ASR is a completely trained ASR file containing all the control characters. TST2.ASR is an ASR file that has not been trained. Both files have the same number of levels with the ASR levels the same. The user may train all the control characters in TST2.ASR using CNTRL.ASR.

```
TRN>TST2/M
TRN>CNTRL/L1        ;   Define ASR file CNTRL, as L1.
TRN>COPY/L1         ;   Using L1, resolve control characters
TRN>L/U             ;   Only control characters are defined,
                    ;   and therefore will not be included in
                    ;   the list.
```

COPY/LIB'N': KEY

This option is used when the user wishes to use only part of LIB'N', to train the current ASR file. Any ASR library in LIB'N' associated with 'KEY' will be compared with the current scope of the ASR file being trained. When using this option, it is not necessary for both files to have the same number of levels or for the ASR levels to be equivalent.

EXIT

When the user wishes to exit the 'TRAIN UTILITY', this command will simply stop the utility.

EXIT/FINISHED

If all the words have been trained, the user can create an ordered ASR file and an ordered SDF. To do this, enter the command EXIT/FINISHED. After creating an ordered ASR file and an ordered SDF, the 'TRAIN UTILITY' will be exited.

At the conclusion of this command, there will be: 1) The ordered SDF, 2) the unordered completely trained ASR file, and 3) the ordered ASR file. The ordered ASR file will have the same filename and extension as the unordered ASR. The version of the ordered ASR will be more recent.

3.3 ERROR MESSAGES

To assist the user in correcting any errors that might occur meaningful error messages have been included. Below is a listing of the error messages and an explanation of what would cause them.

11 - AN ASR ENTRY WAS IDENTICAL TO ZERO

A 32 word voice characteristics entry was completely filled with zeros.

12 - OVER 127 LIB ENTRIES WITH SAME NO BITS SET

There were more than 127 voice characteristic entries which when converted to binary were found to have the same number of bits set. This is not to say that any of the entries were identical.

13 - TWO OR MORE ASR LIB ENTRIES ARE IDENTICAL

There were two or more voice characteristic entries which when compared with each other were found to be identical.

14 - SYNTAX ERROR: ILLEGAL FILE/KEY WORD OPTION

An option chosen by the user does not exist, ie. LIST/ZZZ would generate this error.

15 - SYNTAX ERROR: ABSENCE OF INPUTTED OPTION

The user failed to enter an option.

16 - SYNTAX ERROR: ABSENCE OF KEY ENTRIES

The user failed to enter any key entries when required to.

17 - I/O ERROR: UNABLE TO FIND OR OPEN FILE

The user tried to act on a file that was not in the current directory. The file may also be locked.

18 - I/O ERROR: ASR FILE IN BAD MODE

The user attempted to modify a reordered ASR file.

19 - I/O: LIB LOCATION PREV ALLOL AS STRING LIB

The user tried to define two distinct ASR files with the same library number.

20 - NO ASR DEVICE IN SYSTEM DEFINITION

Train is to be only used when there is an ASR device. If there is no ASR device in the SDF, there is no reason to use the TRAIN UTILITY.

21 - NO EQUATED SDF LIBRARY

A SDF library file was requested which does not exist.

22 - ONLY 1 KEY ENTRY IS ALLOWED

Attempt to enter more than one key entry when only one is allowed.

23 - EXITING ERROR: UNRESOLVED INPUTS REMAINING

The command EXIT/FINISHED was issued before the current ASR file was completely trained. The user may exit by issuing the command EXIT.

4. USER INTERFACE WITH IDE SYSTEM

The purpose of the IDE System is to provide a means of consolidating various data entry devices into a usable package. The IDE task accepts data input from the various devices, and uses the system definition file to find the meaning of a given set of entries.

The user must interface the IDE task in two distinct areas. First, he must write a FORTRAN program which calls two IDE subroutines to request that the IDE task be run and to collect the data (i.e., the definitions). Second, as he runs his program, he will be directly interactive with the IDE task in terms of data entry and control.

4.1 USER CALLING PROGRAM

After creating [and training] a system definition file, the user will enter his FORTRAN program. He will reference the IDE system through calls to the "IDELIB.OLB" subroutine collection. The user program will be of the form shown below.

```
!DATA DECLARATIONS!
!EXECUTABLE STATEMENTS!
      :
CALL INITDE (LIBRAR, LSTDEV, ITOL, IERR)
      :
!EXECUTABLE STATEMENTS!
      :
CALL IDE (BUFFER, NOREQ, NOSNDS, IERR, MODE)
      :
!EXECUTABLE STATEMENTS!
      :
STOP
END
```

There are two subroutine calls to interface between the user task and the IDE task.

4.1.1 INITDE Subroutine

The user first calls "INITDE" to request that the IDE task be run. This routine also sends a command packet to the IDE task containing the system definition file name, the echo device, and the correlation tolerance. Time out on acknowledgement return from the IDE task is handled and an exit handler for user task cleanup is set up. The calling sequence is:

```
CALL INITDE (LIBRAR, LSTDEV, ITOL, IERR)
```

where the arguments are defined as follows.

- LIBRAR (input) is declared to be a 32 BYTE array, and gives the file name for the SDF [and ASR] file[s]. The system has a default extension for the SDF file, ".SDF". Thus, the extension is only required to be a part of the name supplied in LIBRAR when it is not ".SDF". [Note: This default is consistent with the default of the CREATE task]. The ASR library, if applicable, must have the same name as the SDF file and the extension ".ASR". This requirement is consistent with the TRAIN utility creation of the ASR library file.

- LSTDEV (input) is declared to be a 6 BYTE array, and gives the mneumonic name for the listing (or echoing) device to be used for the IDE system messages and data entry echoing capabilities. The mneumonic name may be any that the system has been built to recognize. The syntax for the device name must be a two character alphabetic device name followed by a 0-, 1-, or 2-digit decimal unit number and a colon (ex. TI:, TTO:, LP:). The user may also specify a null device (NL:) if he desires to suppress all echoing messages, although this will hinder his error correction options.

There is an additional restriction on the combined sizes of LIBRAR and LSTDEV. Together these two arrays may not exceed 23 bytes, i.e., 23 alphanumeric characters. Both LIBRAR and LSTDEV should be terminated by a null byte.

- ITOL (input) is declared to be an INTEGER *2 variable specifying the tolerance to be used in the ASR correlation algorithm. In the case where the SDF does not include the ASR device, the tolerance has no significance. For a system including the ASR device, if the tolerance is selected to be too small, the physical limitations of the speech preprocessor, and the user's voice variations will prevent the appropriate match retrieval. If the tolerance is too large, needless time will be spent searching over entries.

- IERR (output) is declared to be an INTEGER *2 variable providing the user with a status return from the INITDE routine. A positive value indicates that the INITDE routine successfully completed. A negative value indicates that an error was detected during the execution of the INITDE subroutine, reported, and immediately returned to the user task.

4.1.2 IDE Subroutine

Before the user begins his interaction with the IDE system in the form of data entry, he will call the IDE subroutine to set his task up to receive data. The calling sequence is

```
CALL IDE (BUFFER, NOREQ, NOSNDS, IERR, MODE)
```

where the arguments are defined as follows.

- BUFFER (output) is declared to be a 26 x NOREQ BYTE array for storing the number of requested definitions. The first 25 bytes of each definition are returned in BUFFER.

- NOREQ (input) is declared to be an INTEGER *2 variable and gi s the numbe: of requested definition retrievals.

- NOSNDS (output) is declared to be an INTEGER *2 variable and returns to the user the actual number of definitions received upon execution of an exit condition (flush command and/or NOREQ satisfied) for the IDE subroutine.

- IERR (output) is declared to be an INTEGER *2 variable and reports the user status. If IERR is positive, the IDE subroutine completed successfully. If IERR is negative, an error condition was encountered during the execution of the IDE subroutine, was reported to the user, and the IDE subroutine returned immediately to the calling routine.

- MODE (input) is declared to be an INTEGER *2 variable and specifies one of two possible modes of operation. MODE = 0: The purpose of this mode is to provide the user with immediate access to the definitions returned from the IDE task. The normal exiting condition is the satisfaction of the number of requested definitions. However, all of the special control characters of the IDE task may also be used. (These control characters will be discussed in detail in the Section titled "Operational Features of the IDE Task").

  MODE $\neq$ 0: For any value other than 0, the mode selected will provide the user with completely updated (corrected) data. The error control command is handled by this mode until that time when a flush control command is entered. Any definitions retrieved beyond the number requested are discarded and the user is warned, unless that retrieval is a correction on the previous definition as indicated by an error control command. The only means of returning to the user task is by issuing the flush command. The user is prompted by the IDE task when the number of requested definitions has been received and that it is an appropriate time to flush if he is satisfied. The user may, however, use the flush command an·

time before the last requested definition has been received, as well, to terminate the session prematurely.

Once the user has received the data via the IDE subroutine, he may use any executable FORTRAN statements to manipulate, report, or make use of the data to serve his purposes. He may make repeated calls to the IDE subroutine within his program. When the user terminates his FORTRAN program, the IDE task will also be terminated.

The user program should be compiled and then linked with the IDE library, IDELIB.OLB, during task building.

4.2 OPERATIONAL FEATURES OF THE IDE TASK

Once the user has written his FORTRAN program and is ready to run his task, he will be interacting with the IDE task.

4.2.1 Pre-Run Time Initialization of the IDE Task

4.2.1.1 IDE Installation

The user should install the IDE task. If he is using the ASR device, he has an option of two correlation algorithms. The first algorithm makes one comparison between the spoken word and each ASR library entry within the specified tolerance. The second algorithm makes three comparisons (one bit by bit and two shifts) between the spoken word and each ASR library entry within the specified tolerance. The task images are IDEALG1 and IDEALG2 and the user may make his choice at installation time. The first algorithm favors speed of correlation while the second algorithm favors matching under slight user speech variations.

4.2.1.2 Device Assignment

Each of the devices in the system definition file requires its own logical unit number. The user should verify that the devices in his system definition are physically ready (i.e., power on, etc.). The user should assign the appropriate logical unit numbers to the devices in his system definition by making assignments to the pseudo devices, DV1:, DV2:, DV3:, as follows.

```
> ASN     Device 1  mneumonic = DV1:
> ASN     Device 2  mneumonic = DV2:
> ASN     Device 3  mneumonic = DV3:
```

The device order is dictated by the system definition.

For example, consider the system definition: TT0, TT1, ASR where TT0 is the physical device TT2:, TT1 is the physical device TT3:, and ASR is the physical device TT7:. Then, the user would make the following assignments.

```
> ASN     TT2:=DV1:
> ASN     TT0:=DV2:
> ASN     TT7:=DV3:
```

4.2.2 Run Time

When the user runs his task, the "CALL INITDE ( )" will request that the IDE task be run and the "CALL IDE ( )" will complete the send of data to the IDE task which is required for it to be functional. The IDE task will be in a wait for data entry state, and when it has arrived at this point, it will prompt the user on the echoing device with:

```
** READY FOR DATA ENTRY **
```

This prompt will be repeated each time the user task executes "CALL IDE ( )". At this point, the user may begin his data entry abiding by the following

rules.

4.2.2.1 Data Entry

The integrated data entry system is based on receiving the inputs from the devices in the order specified in the system definition file. There are three exceptions to this rule.

a. A control character may be entered from any device. (The control characters will be discussed in the following subsection).

b. Any of the higher order devices (that is, devices before the last device in the system definition file) may omit a device input with the default "entry" being the entry for that particular device from the previous set.

For example, the following two sets of data entry would yield the same results.

DV1	DV2	DV3	DV1	DV2	DV3
A	1	X	A	1	X
(omit)	2	X	A	2	X
(omit)	(omit)	Y	A	2	Y
B	(omit)	Z	B	2	Z

This option saves the user from redundant entries of the higher order devices.

c. Any of the higher order devices may be corrected by resubmitting that entry any time prior to entering the lowest device entry. For example, consider the following sets of data input.

	CORRECTED INPUT		
	DV1	DV2	DV3
(DV1) A, (DV1) B, (DV2) 1, (DV3) X	B	1	X
(DV1) A, (DV2) 1, (DV1) B, (DV3) X	B	1	X
(DV1) A, (DV2) 1, (DV2) 2, (DV3) X	A	2	X

The IDE system is currently built to handle spoken words from the ASR device, and entries of size one alphanumeric character from the terminals and the DCR device. The IDE system will allow the user to work ahead (up to 4 entries per device, not to exceed 10 in total) by continuous entry of data, as long as the data entry is in accordance with the rules above. However, this work ahead option may hinder error correction.

4.2.2.2 Special Control Characters

The special control characters may be entered on any device in the system definition file, and at any time with the exception that an ASR control may only be entered after at least one entry for the key devices has been previously entered. (An ASR library must be initialized). There are three special control characters and a description of their use will follow.

^C (control C) - ABORT/TERMINATE/KILL

Issuing the abort command will terminate the IDE and the user tasks, after task cleanup. Control will return to the MCR command line.

^U (control U) - ERROR CONTROL/CORRECTION

The error control command is issued after an erroneous definition has been sent to the user task. The interfacing subroutines handle the error control by replacing the previous entry with the current, corrected entry. There is no limit on the number of times an entry may be corrected by using

the error control, however, only the most recently sent definition may be corrected. The use of ERROR CONTROL is the only means of correcting the lowest device input, as well as the set of inputs, i.e., the definition. For MODE=0, the ERROR CONTROL may be used on all but the last requested definition retrieval (this restriction is imposed on MODE=0 since as soon as the NOREQ has been satisfied, the IDE subroutine returns to the calling program). For MODE≠0, the error control may be used on all retrievals.

^W (control W) - FLUSH/FINISH

In MODE≠0, the flush command is the only method of ending the session, returning the retrieved definitions from the IDE subroutine to the user's calling program. In either mode, the user may choose to enter the flush command any time prior to receiving the last requested definition. The current number of definitions, and a count of this number will be returned to the user's calling program.

4.2.3 Echoing Features and Messages

As mentioned above, each calling to the IDE subroutine will cause a prompt to be displayed indicating that the IDE task is waiting for data input:

** READY FOR DATA ENTRY **

After each entry is submitted, the mneumonic for the entry device and the entry are echoed to the user on the LSTDEV, for the user's inspection and approval or correction. In the case of an ASR entry, the spoken word is first correlated against the ASR library, and if an acceptable match is found, the meaning associated with that spoken word is what is echoed to the user. Once an entry from the lowest device has been received, the definition associated with the given set of inputs is found and echoed to the user and sent to the user's task. The format of the echoing is designed for easy comparison with the SDF listing option of the CREATE task and has the following form.

```
DV1     xxx
        DV2     xxx
                DV3     xxx
                        =xxxxxxxx
```

Also echoed to the user are messages related to the control characters.

^C:	** IDE TERMINATED **
^U:	DVn ** ERR CTL **
^W:	DVn ** FLUSH **

During data entry sessions there are three error messages for the user which directly relate to his data entry and are also reported on the echo device, LSTDEV.

a. 33- NO ASR LIBRARY MATCH WITHIN TOLERANCE

The spoken word has been correlated against all possible ASR entries within the given tolerance and no match was found. The user may repeat the word. If this error occurs too frequently, the tolerance might be increased, or the user might check his source system definition file to verify the existence of this word in the ASR library.

b. 47- ASCII INPUT -- NO MATCH IN SDF ON PATH

No match in the SDF for the ascii input has been found based on the device the entry was received on and the inputs for the devices before this entry. The user should examine his system definition file.

c. 49- WARNING: INPUT DISCARDED // FLUSH OR ERR CTL

This message will only occur in MODE≠0. When the user attempts to submit a set of entries after the last requested definition has been received, and it has not been preceded by an error control command, this

superfluous definition is discarded.

In ending the data entry session, there are prompts reporting the end to the user. For MODE≠0, when the last requested definition has been sent, the user receives the message:

** LAST ENTRY SENT: FLUSH IF CORRECT, ELSE ERR CTL AND RESUBMIT **

This message is repeated each time the user enters ERR CTL and resubmits his set of data. For MODE=0, when the last requested definition has been sent, and the IDE subroutine returns the definitions to the user's program, the following message is echoed:

** LAST REQUESTED ENTRY SENT **

Any time the user enters the flush command, either to end a session prematurely or to end a session of MODE≠0 normally, the system prompts the user with:

** IDE COMPLETED **

For an example of the echoing features of the IDE system, consider one call of the IDE subroutine with MODE≠0 and NOREQ=3.

```
    ** READY FOR DATA ENTRY **
    TTO   C
        DCR   C
            ASR  MONZA
               =  MONZA
            ASR  MONZA
               =  MONZA
ASR ** ERR CTL **
          ASR  CAPRICE
               =  CAPRICE
TTO    B
          ASR  OWENS
               = OWENS
** LAST ENTRY SENT: FLUSH IF CORRECT, ELSE ERR CTL AND RESUBMIT **
DCR ** ERR CTL **
              ASR   LUHERS
                    = LUHERS
** LAST ENTRY SENT: FLUSH IF CORRECT, ELSE ERR CTL AND RESUBMIT **
TTO ** FLUSH **
** IDE COMPLETED **
```

The user program would have stored in BUFFER:

MONZA

CAPRICE

LUHERS

4.2.4 Error Conditions

During the operation of the IDE task, there are 17 possible error messages the user may encounter. A description of these errors, their possible sources and remedies is included here. The three errors discussed above in the section on echoing are the three most common during run time. Many of these other errors should not occur if the SDF and ASR files have been successfully built.

17 - I/O ERROR: UNABLE TO FIND OR OPEN FILE.

An SDF file has been requested which does not exist, or is locked. The user should check the filename and extension.

31 - I/O ERROR: FCS CODE VIOLATION DURING READ

32 - I/O ERROR: END OF FILE REACHED DURING READ

These two errors are traps to prevent violations during reading the ASR file. The user should verify the existence of the ASR file and that his training is complete.

33 - NO ASR LIBRARY ENTRY MATCH WITHIN TOL

The correlation algorithm returns no acceptable match to the word spoken. The user may repeat the word, check the tolerance, use IDEALG2, verify the existence of this word in the SDF file, and/or check his training of this word.

34 - INPUT ERROR: QUEUE ALREADY FULL

Each device queue allows the user to work up to 4 entries ahead, with a total work ahead ability of 10 entries for all devices. This error indicates that the user has exceeded these limits.

35 - OUTPUT ERROR: QUEUE ALREADY EMPTY

This error indicates that the system was requested to retrieve an entry from the device queue, when there was no entry present.

40 - DEVICE NOT IN SYSTEM DEFINITION

An entry has been received from a device not in the system definition or the SDF contains a device mneumonic which is not one of the five recognized: TT0, TT1, TT2, DCR, ASR.

41 - RECEIV DIRECTIVE FAILURE

43 - SEND DIRECTIVE FAILURE

50 - ERROR FOR REC DATA AST (IN FLGREC)

These three errors trap for errors in the sending and receiving of the command and data packets. There is a five second time out safeguard, and if this time has been exceeded an error will be reported.

42 - REQUEST DIRECTIVE FAILURE

The request that IDE be run has failed. The user should verify that the IDE task has been installed.

44 - ASR LIBRARY HAS UNRESOLVED INPUTS

An attempt to run the IDE task with an incomplete version of the ASR library has been attempted. The user should first complete the TRAIN utility session.

45 - SDF FILE DOES NOT HAVE DEVICE LIST

The first record of the SDF file is not the device record, as expected. The user should examine his SDF file.

46 - PATH SPECIFIED DOES NOT EXIST IN SDF

The path given by the ascii inputs is invalid for the SDF. The user should verify that his entries are consistent with the system definition.

47 - ASCII INPUT -- NO MATCH IN SDF ON PATH

The current entry has no match in the SDF based on the ascii text and the entries for the higher-order devices.

48 - DEVICE INITIALIZATION FAILURE

During initialization of the devices in the system definition file, an error was detected. The user should verify that all of the devices in his system definition are ready to be connected to the system.

49 - WARNING: INPUT DISCARDED // FLUSH OR ERR CTL

In MODE ≠ 0, a set of inputs beyond the number requested has been entered without specifying ERR CTL. The set of inputs is discarded, and the system waits for a flush or error control command.